-- UnifiedFonts.mesa Unified Font Interface

```
--MichaelPlass,October21,19821:06pm

DIRECTORY Rope,Graphics;

UnifiedFonts:CEDAR DEFINITIONS =
  BEGIN
  ROPE: TYPE = Rope.ROPE;
  Context:TYPE = Graphics.Context;
```

This interface is designed to be used in conjunction with CedarGraphics, and perhaps someday it will become integrated with that package.

```
  FONT: TYPE = REF FontRec;
```

A FONT describes a font, that is, a collection of related graphical objects. Each object, or character, in the collection is referred to by an index of type CHARACTER. There are various attributes of a font that may be accessed through this interface; some of them pertain to the font as a whole, and others pertain to the individual characters.

This interface is designed to be used in conjunction with CedarGraphics, and perhaps someday it will become integrated with that package.

The actual graphical objects contained in a font may or may not be locally available. For hardcopy applications, the exact graphical descriptions typically reside at the print server, and cannot be assumed to be available. On the other hand, the graphical descriptions for fonts to be used on the screen must be locally available. In any case, some sort of reasonable approximation is attempted for local use.

A font is ideally available in all sizes and rotations; however, not all print servers are capable of living up to this ideal. Therefore, a mechanism is provided for finding out what transformations are available (easy) for a font on a particular (kind of) output device.

*The association between CHARACTER codes and the images is often jumbled, due to historical reasons and the natural slowness in the propagation of standards. As a step towards untangling this mess, this interface provides a way to find out the **Xerox Character Code** for a character in a given font, and vice versa.*

```
  FromName: PROCEDURE [fontName:ROPE] RETURNS [FONT];
```

This gets at least the font metric information from a centralized database. The font name is an Interpress hierarchial font name, and encodes all of the family, face, and design size information. The font will be in a size designed for one-unit baseline spacing, which is probably not right for most applications.

```
  Name: PROCEDURE [font:FONT] RETURNS [fontName:ROPE];
```

This extracts the name of the font. May return NIL if the font is unnamed.

```
  Transformation: TYPE = RECORD [m11,m12,m21,m22: REAL];
```

*A Transformation specifies the matrix for the linear transformation used to scale, rotate, and skew fonts. **Scale** and **Rotate** are useful for building up the most common transformations.*

```
  Rotate: PROCEDURE [transformation:Transformation_identity, degrees:REAL] RETURNS
    [Transformation];

  Scale: PROCEDURE [transformation:Transformation_identity, magnification:REAL] RETURNS
    [Transformation];

  identityTransformation:Transformation=[1.0,0.0,0.0,1.0];

  ModifyFont: PROCEDURE [font:FONT,transformation:Transformation];
```

This applies the given transformation to the font, to make it be the desired size and orientation. This is equivalent to applying the transformation to the display context before the characters are displayed.

CurrentTransformation: PROCEDURE [font: FONT] RETURNS [transformation: Transformation] INLINE {RETURN [font.currentTransformation]};

Useful for finding out how the font has been transformed.

DeviceType: TYPE = ROPE;

Tells what kind of device to tailor the easy transformations for. "Ideal" as a device type will allow any transformation. The name of a print server may function as a DeviceType, when the implementation is smart enough to allow this.

EasyTransformations: PROCEDURE [font: FONT, deviceType: DeviceType, transformationProc: TransformationProc];

TransformationProc: TYPE = PROCEDURE [transformation: Transformation] RETURNS [quit: BOOLEAN _ FALSE];

anyTransformation: Transformation = [0,0,0,0];

EasyTransformations *enumerates the transformations of the font that are easy on the specified device. If any transformation is allowed,* ***anyTransformation*** *will be included in the list of easy transformations. The transformations are all expressed relative to the font as currently modified.*

SubstituteForDevice: PROCEDURE [font: FONT, deviceType: DeviceType, transformation: Transformation];

Changes the font to one that is available on the specified device. An attempt will be made to make a reasonable substitution.

Contains: PROCEDURE [font: FONT, char: CHAR] RETURNS [BOOLEAN];

TRUE iff the character exists in the font.

NSCode: PROCEDURE [font: FONT, char: CHAR] RETURNS [nsCode: INT];

unassignedNSCode: INT = 255;

nonexistentNSCode: INT = 65535;

Returns the Network Systems code for the specified character; ***unassignedNSCode*** *is returned for characters that have no such code assigned, and* ***nonexistentNSCode*** *is returned for characters that do not exist in the font.*

CharCode: PROCEDURE [font: FONT, nsCode: INT] RETURNS [char: CHAR];

Returns the CHARACTER code for the specified Network Systems code; NUL is returned for characters that do not exist in the font.

DrawChar: PROCEDURE [font: FONT, char: CHAR, context: Context];

Displays a single character through Graphics.

Font Metrics

FontMetricCode: TYPE = {xHeight, slant, underlineOffset, underlineThickness};

GetFontMetric: PROCEDURE [font: FONT, characterMetricCode: CharacterMetricCode] RETURNS [REAL];

Character Metrics

CharacterMetricCode: TYPE = {widthX, widthY, leftExtent, rightExtent, descent, ascent, centerX, centerY, superscriptX, superscriptY, subscriptX, subscriptY};

GetCharacterMetric: PROCEDURE [font: FONT, characterMetricCode: CharacterMetricCode, char: CHAR] RETURNS [REAL];

Kern: PROCEDURE [font: FONT, char1, char2: CHAR] RETURNS [REAL];

Returns the amount to add to the character's width to kern this character to the successor character.

Ligature: PROCEDURE [font: FONT, char1, char2: CHAR] RETURNS [CHAR];
Returns NUL if there is no ligature for the two characters.

Derived Character Metrics

The following procedures access some metrics that are useful when the full generality of the GetCharacterMetric procedure is not needed. Access through these procedures will be speedier than through the general machanism.

Width: PROCEDURE [font: FONT, char: CHAR] RETURNS [REAL];
Returns the magnitude of the width vector.

PosExtent: PROCEDURE [font: FONT, char: CHAR] RETURNS [REAL];
Returns the magnitude of the maximum extent of the character in the direction 90 degrees counterclockwise to the width vector. For unrotated Latin alphabets, this is the Interpress ascent or the TEX height.

NegExtent: PROCEDURE [font: FONT, char: CHAR] RETURNS [REAL];
Returns the magnitude of the maximum extent of the character in the direction 90 degrees clockwise to the width vector. For unrotated Latin alphabets, this is the Interpress descent or the TEX depth.

Press Compatibility Stuff

```
PressFontSpecification: TYPE = RECORD [
    family: ROPE,
    face: PressFontFace,
    size: INTEGER,
    rotation: INTEGER
    ];

PressFontFace: TYPE = [0..256);

GetPressFontSpecification: PROCEDURE [FONT] RETURNS [PressFontSpecification];
```

Implementor Private Stuff

```
FontRec: TYPE = PRIVATE RECORD [
    name: ROPE,
    currentTransformation: Transformation,
    bc, ec: CHAR,
    nsCode: NSCodeRec,
      --some stuff goes here for Width, PosExtent, NegExtent, Kern, and Ligature.
    DrawCharProc: PROCEDURE [font: FONT, char: CHAR, context: Context],
    data: REF ANY
    ];

NSCodeRec: TYPE = PRIVATE RECORD [
    code: SEQUENCE length: NAT OF CARDINAL
    ];

END.
```

```
INLINE bodies are down here to avoid clutter until the interface looks right.
    Contains = INLINE {RETURN [NSCode[font, char] # nonexistentNSCode]};
    NSCode = INLINE {RETURN [IF char>=font.bc AND char<=font.ec THEN
    font.nsCode[char-font.bc] ELSE nonexistentNSCode]};
    CharCode = INLINE {RETURN [IF nsCode<255 AND NSCode[font, 0C + nsCode]=
    nsCode THEN 0C + nsCode ELSE FindCharCode[font, nsCode]]};
```